

Real-time Operating Systems and Systems Programming

Scheduling

Classical scheduling

- Two goals:
 - Maximize processor usage
 - Minimize response time of tasks
- Evaluation:
 - Task waiting time
 - Processor throughput
 - Total execution time of tasks
 - Average response time of tasks

Scheduling decisions

- Preemptive or non-preemptive
- Static or Dynamic
- Soft or Hard (Best effort vs Strict)

Scheduling strategies

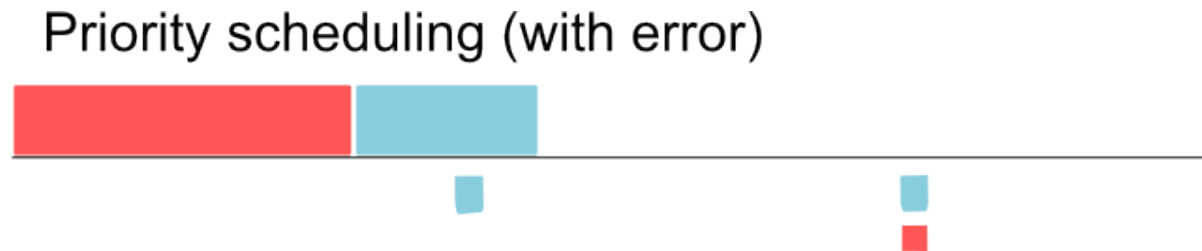
- Round-robin
- First come first served (FIFO queue)
- Prioritized scheduling
- Deadline prioritization
- Shortest first

Implementation details

- Election table
- Priority queue list
- For complex scheduling, *two level* scheduling can be implemented
 - High level decisions on general policy that affect longer periods
 - Lower level scheduler decides reordering for immediate future

Priority scheduling

- Red is of higher priority, but with longer deadline



Rate monotonic scheduling

- Priority is inverse of the period
 - Short periods are high priority and vice versa

Rate monotonic scheduling



Rate monotonic scheduling with error



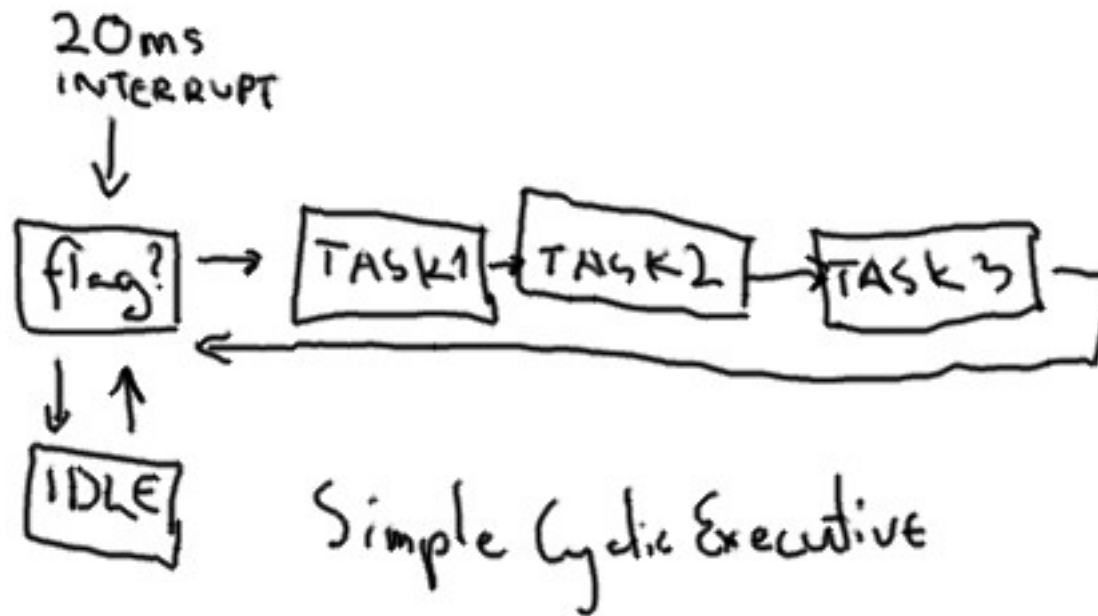
Earliest deadline first

- Priorities are assigned according to deadlines

Earliest deadline first scheduling



Cyclic Executive



Cyclic Executive

- Offers 3 priorities for tasks
- Interrupt if pre-emption is needed (high priority)
- HW Clock with cooperative scheduling (middle priority)
- Base code with first come first served priority (low priority)
- Inflexible, since does not offer task “aging” or dynamic scheduling
- Simple to test

Tasks exceeding time-slot

- Might be possible to split larger tasks.
- Starts on even ticks, then yields time to others
- Finishes on odd ticks

Idle

- Idle task could be replaced with low-priority things
- Also called Burn

Posix scheduling

- Pthread scheduling with function:
 - sched_setscheduler()
 - SCHED_FIFO – realtime
 - SCHED_RR – realtime with timeslots
 - SCHED_OTHER – normal scheduling
 - SCHED_BATCH – less prioritized normal (Linux 2.6.16+)
 - SCHED_IDLE – lowest possible priority (Linux 2.6.23+)

SCHED_FIFO

- FIFO processes pre-empt any OTHER and BATCH processes.
- If FIFO process is pre-empted it will resume as soon as higher-priority processes are blocked
- If becomes runnable, inserted to queue
- `sched_setscheduler()` puts in front of queue if runnable (ignoring POSIX)
- `sched_yield()` to send self to end of list

SCHED_RR

- Round Robin enhances FIFO
- Each process gets maximum time quantum
- If runs longer, put to end of queue
- `sched_rr_get_interval()` will return the quantum

SCHED_OTHER

- Normal way of things
- Processes run according to the nice value
 - `nice()` or `setpriority()` used to set the value
- The priority increases each quantum processes are ready, but denied time
- SCHED_BATCH assumes CPU-intensive process which is not interactive and it gets a penalty in priority

Permissions

- CAP_SYS_NICE permission needed
- Unprivileged processes can set SCHED_OTHER for the same user
- Can be overridden
- Processes running under SCHED_IDLE cannot change to something other without permissions

Miscellaneous

- Child processes inherit their parents' scheduling policy
- Real-time processes need memory locking (`mlock()` and `munlock()`) to avoid paging delays
- A non-blocking loop in `_FIFO` or `_RR` priority will lock the computer unless a shell is scheduled on same level prior to running it for killing it off. Remember when debugging.

Process creation and destruction

- Unix offers 4 system calls for process creation, destruction and waiting for them to finish:
 - `exec()` family
 - `fork()`
 - `wait()`
 - `exit()`

Loading of a process

- Binary executable contains header, (program) text, data, relocation information and symbol table. Text and data will be loaded with program

Executable file

Process memory

HEADER

TEXT

TEXT (program)

DATA

DATA (initialized)

(BSS)

BSS (=uninitialized data)

free mem

RELOCATION

STACK

SYMBOL TABLE (can be stripped)

USER BLOCK (in kernel adr space)

exec() family

- Exec loads a binary executable into memory and starts a process.

```
extern char **environ;  
int execl( const char *path, const char* arg, ...);  
int execv (const char *path,  char *const argv[]);  
int execl( const char *path,  
           const char *arg, ..., char * const envp[]);
```

- execl : full file path, arguments as chars
- execv : full file path, arguments as array
- execl : full file path, arguments as chars, environment

Environment

- `getenv()`
- see also `putenv()`

exec() family (2)

- The real function is execve()

fork()

wait()

waitpid()

system()

atexit()

Demon

Zombie

Process